

Deep Learning of Robotic Tasks using Strong and Weak Human Supervision

Bar Hilleli & Ran El-Yaniv
 Department of Computer Science
 Technion - Israel Institute of Technology
 Haifa, Israel
barh@campus.technion.ac.il
rani@cs.technion.ac.il

Abstract

We propose a scheme for training a computerized agent to perform complex human tasks such as highway steering. The scheme resembles natural teaching-learning processes used by humans to teach themselves and each other complex tasks, and consists of the following four stages. In the first stage the agent learns by itself an informative low-dimensional representations of raw input signals in an unsupervised learning manner. In the second stage the agent learns to mimic the human instructor using supervised learning so as to reach a basic performance level; the third stage is devoted to learning an instantaneous reward model. Here, the (human) instructor observes (possibly in real time) the agent performing the task and provides reward feedback. During this stage the agent monitors both itself and the instructor feedback and learns a reward model using supervised learning. This stage terminates when the reward model is sufficiently accurate. In the last stage a reinforcement learning algorithm is deployed to optimize the agent policy. The guidance reward signal in the reinforcement learning algorithm relies on the previously learned reward model. As a proof of concept for the proposed scheme, we designed a system consisting of deep convolutional neural networks, and applied it to successfully learn a computerized agent capable of autonomous highway steering over the well-known racing game *Assetto Corsa*.

1 Introduction

Consider the task of designing a robot capable of performing a complex human task such as dishwashing, driving or clothes ironing. Although natural for adult humans, designing a hard-coded algorithm for such a robot can be a daunting challenge. Difficulties in accurately modeling the robot and its interaction with the environment, creating hand-crafted features from the high-dimensional

sensor data, and the requirement that the robot be able to adapt to new situations are just a few of these obstacles. In this paper we propose a general scheme that combines several learning techniques that might be used to tackle such challenges. As a proof of concept, we implemented the scheme’s stages (currently without the initial unsupervised learning stage) and applied it to the challenging problem of autonomous highway steering. The implementation of the unsupervised learning stage is currently being performed and will be added in a future version of the paper.

Two of the most common approaches for robot learning are *Imitation Learning* (IL) and *Reinforcement Learning* (RL). In imitation learning, which is also known as ‘behavioral cloning’ or ‘learning from demonstrations’, a human demonstrator performs the desired task with the goal of teaching a (robotic) agent to mimic her actions (Argall et al., 2009). The demonstrations are used to learn a mapping from a given world state, s , received via sensors, to a desired action, a , consisting of instructions to the agent’s controllers. Throughout this paper s will be referred also as the “raw signal.” The objective in IL is to minimize the risk, in the supervised learning sense. In RL the goal is to enable the agent to find its own policy, one that maximizes a value function defined in terms of certain guidance reward signals received during its interaction with the environment. IL and RL can be combined, as was recently proposed by Taylor et al. (2011). The idea is to start the reinforcement learning with an initial policy learned during a preceding IL stage. This combined approach can significantly accelerate the RL learning process and minimize costly agent-environment interactions. In addition, deploying the RL algorithm after the IL stage compensates for the noisy demonstrations and extends the imitation strategy to previously unseen areas of the state space.

Both IL and RL have their drawbacks. IL can suffer from noisy demonstrations, which typically result from inconsistent strategies adopted by the human demonstrator. Consequently, the performance of an IL agent is limited by the quality of the observed demonstration. RL has its drawbacks as well: it either requires a realistic simulation of the agent’s interaction with the environment, or it requires operating the agent in a real-world environment, which can be quite costly. Moreover, the sample complexity of RL can be large and the modeling of an effective reward function is often quite challenging, requiring expert insight and domain knowledge. The current state of affairs in robotic design using any technique leaves much to be desired, where ultimate goals such as domestic robot maids, remain futuristic.

Consider the task of constructing a robot (or any computerized agent) whose goal is to perform a certain task based on raw signals obtained via sensors. The sensors can be of several types and modalities, such as sound, imaging, proximity (sonar, radar), navigation, tactile, etc. For the moment, we ignore the means by which we process and integrate the various signals, and simply refer to the given collection of signals as the “raw signal.” We propose the following scheme that integrates unsupervised feature learning, imitation learning, reward shaping, and reinforcement learning as follows:

1. **Unsupervised learning.** Utilizing known unsupervised learning techniques the agent learns informative low-dimensional representations $F_0(s)$ of the raw signal s . Typically, the representation is hierarchical, in the form of artificial neural network whose inputs are obtained from the raw signal and the output is the lower dimensional representation.
2. **External features.** In many tasks, such as autonomous driving, there are high level features of the raw signal that are known to be relevant to the task and accelerate the learning of the robot. For example, road boundaries, which can be easily extracted using simple image processing algorithms (e.g., the Hough Transform (Illingworth & Kittler, 1988)), are of utmost relevancy to the task. Such engineered features that are based on domain expertise should definitely be used whenever they exist to expedite learning and/or improve the final performance. In addition, other auxiliary features that are not directly learned by our system, or engineered by us, and can obtained as black boxes from professional feature manufacturers can be considered. Any set of such *external features* can be easily integrated into our scheme using known methods such as those described in (Ngiam et al., 2011).
3. **Supervised imitation learning.** Using the learned low-dimensional (hierarchical) representation $F_0(s)$ (comprised of both self-learned features and possible external features) to initialize an imitation learning process, the agent learns to mimic an (human) instructor performing the desired task. This stage has two complementary goals. The first goal is to generate an initial agent policy π_0 capable of operating in the environment without too much risk (e.g., without damaging itself or the environment). The second goal is to improve the low-dimensional feature representation (learned in the previous unsupervised stage) and generate a revised representation $F_1(s)$, which is more relevant/informative to the task at hand.
4. **Supervised reward shaping.** The agent *learns* a reward function $R(s)$ (to be used later by the RL procedure) from instructor feedback generated while observing the agent operating in the environment using the initial IL policy π_0 . The reward function learning process utilizes the learned representation, $F_1(s)$, to accelerate learning.
5. **Reinforcement learning.** Finally, in the RL stage, the reward function $R(s)$ is used to learn an improved agent policy π^* . This learning in this stage can be based on the learned representation $F_1(s)$. The representation $F_1(s)$ can remain “frozen” through the RL procedure or it can be updated to better reflect new scenes. Also, the RL stage can remain active indefinitely.

We note that the first unsupervised feature learning stage is not mandatory, but can potentially accelerate the entire learning process and/or lead to enhanced overall performance. When this stage is not conducted, we start with the supervised imitation learning stage from scratch, without $F_0(s)$, and when the

supervised learning process ends, we extract from it both $F_1(s)$ and the initial robot policy π_0 . When the unsupervised feature learning stage is not conducted, the IL stage can take longer time and/or require more efforts from the human demonstrator. We also note that stages three and four can be iteratively repeated several times to improve final performance.

The proposed approach somewhat resembles the natural teaching-learning procedure used by humans to teach themselves and each other complex tasks. For example, in the case of learning to drive, the student’s unsupervised learning phase starts long before her formal driving lessons, and typically includes great many hours where the student observes driving scenes while sitting as a passive passenger in a car driven by her parents during her childhood. In the second stage while observing the instructor performing a desired task, the student extracts relevant information required to successfully perform the task. Afterwards, while the student is performing the task, the instructor provides real-time feedback and the student improves performance by both optimizing a policy as well as learning the feedback function itself. Then, the student continues to teach herself (without the instructor), using both the reward function previously induced by the instructor and future reward signals from the environment.

There is vast literature concerning the independent use of each of the components in our scheme. See, for example surveys on unsupervised feature learning Coates et al. (2010), imitation learning (Argall et al., 2009) and reinforcement learning (Kober et al., 2013). The two closest works to ours are (Taylor et al., 2011) and (Daniel et al., 2014). In the first, the authors showed that a preceding demonstration learning stage can significantly expedite the reinforcement learning process and improve the final policy performance in a simulated robot soccer domain. In the second, the authors proposed to learn a reward model in a supervised manner and used alternating steps of reward and reinforcement learning to continually improve the reward model and the agent’s policy in a robotic arm grasping task. To the best of our knowledge, there are no reports on previous attempts to leverage human instructor’s skills to create robots using the above procedure (with or without the unsupervised learning stage, and with or without external features).

As a proof of concept, we focus on learning a simple autonomous highway steering task and report on an implementation of the proposed scheme, including the imitation learning, reward shaping and reinforcement learning stages (without the unsupervised learning stage, and without external features, which will be added in a future version of the paper.) To this end, we use one of the most popular computer racing games (Assetto Corsa)¹, and attempt to create a self-steering car in the sense that, given raw image pixels (of the car racing game screen), we wish to output correct steering control commands for the steering wheel. We use Convolutional Neural Networks (CNNs) as mapping functions from high-dimensional data to both control actions and instantaneous reward

¹At the time of writing, this game is considered to be one of the most realistic computer racing environments.

signals. Three types of CNNs are trained: policy network, reward network and Q-network, denoted by P_θ , R_θ and Q_θ , respectively. The choice of CNNs for all three tasks is based on their proven ability to extract informative features from images in the context of classification and control tasks (Mnih et al., 2015; Krizhevsky et al., 2012), thus obviating the exhausting task of manually defining features. For example, in the work of Mnih et al. (2015) a CNN was successfully trained to predict desired control actions given high-dimensional pixel data in the Atari 2600 domain.

We emphasize that our entire system is implemented *without* any access to the internal state of the game simulator (which is a purchased executable code). This is in contrast to most previous published work on computerized autonomous driving that were conducted in a simulation environment, allowing access to the internal states of the simulator (e.g., TORCS (Wymann et al., 2000)) containing valuable parameters such as the car’s distance from the road-side or its angle with respect to the road. Thus, when an open simulator is available, such parameters can be extracted and utilized in the learning process, as could other reward information (e.g., in computer game simulators) (Zhang & Cho, 2016; Loiacono et al., 2010; Munoz et al., 2009; Chen et al., 2015). As mentioned, there is no open simulator in our setting. Consequently, in addition to not having the access to useful parameters, all our learning procedures, including the reinforcement learning, must be executed slowly in real-world clock time (as in real driving), as opposed to the super-fast learning that can typically be achieved using a simulator. On the other hand, our system is scalable to any computerized driving game and even to real-world driving.

Our work is divided into three main parts. First, the actions of a human demonstrator are recorded while she plays the game. The game images are recorded as well. A policy network is trained in a SL manner using this data. Second, a reward network is trained. It receives an image and outputs a number in the range $[-1, 1]$ that indicates the instantaneous reward of being in that state. This method is known as *reward shaping*, where an additional reward signal is used to guide the learning agent (Ng et al., 1999). The reward network is trained in a SL manner with labels obtained from a human instructor. Third, the Double Deep Q-learning (Hasselt, 2010; Hasselt et al., 2016) (DDQN) RL algorithm is used to train a Q-network. The reward signal used in the RL procedure is constructed from the reward network’s output. The learned policy network’s parameters from the imitation part are used to initialize the Q-network’s parameters. Our performance evaluation procedure is based on the Q-network’s accumulated average reward. Better driving means higher accumulated average reward and vice versa.

2 Imitation learning

Imitation learning, which is also known as behavioral cloning or learning from demonstrations, aims at finding a mapping $f^* : s \rightarrow a$ from a given world state, s , to a desired action a . This mapping is typically termed “policy.” Robot

learning using mimicry is an old idea, conceived decades ago (Hayes & Demiris, 1994; Argall et al., 2009). While being an excellent technique for achieving reasonable performance, this approach by itself is limited. Clearly, the performance of an agent trained in this manner is upper bounded by the instructor level. Moreover, if the training sample isn’t sufficiently diverse/representative the agent will not be exposed to unexpected difficult states, and can suffer from very poor and unpredictable performance when such states are encountered in production. Finally, labeled samples obtained from human demonstrations are prone to labeling noise. Noting these limitations, we use imitation learning in our scheme only to achieve a basic performance level, which will allow the agent to perform the required task without damaging itself or the environment, and to receive a low-dimensional feature representation of the raw signal, s , that will be used to initialize both the reward learning and the reinforcement learning stages. In our setting, world states are game images, and actions are keyboard keys pressed by a human demonstrator while playing the game. The Assetto Corsa game provides the possibility of connecting a car wheel controller; we have not utilized this option and note that the use of such a wheel controller should improve the driving performance of the learned agent.

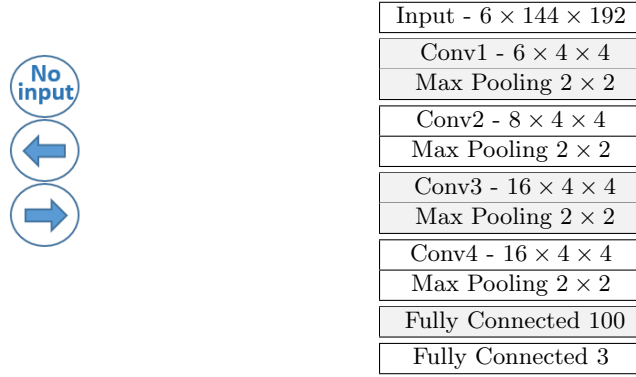
In this stage of the scheme we train a policy network that maps raw image pixels to steering commands of left/right. The policy network, whose architecture is presented in Figure 1, is trained and evaluated. When training is done, the last convolutional layer’s output can be viewed as the learned low-dimensional representation, $F_1(s)$, of the raw signal. A training sample of state-action pairs is gathered in the following two-stage procedure, which can be viewed as one iteration of the *Dagger* algorithm applied with $\beta = 0$ (Ross et al., 2011). The Dagger algorithm was selected to increase the robustness of the resulting mimicking policy.

First, a training sample of state-action pairs, $D_0 = \{(s_i, a_i)\}_i$, is recorded while a human expert plays the game. A detailed explanation of the recording process is given in Section 5 (in the experiments described below D_0 contained approximately 70,000 samples, equivalent to two hours of human driving). Denote by P_{θ_0} the learned policy network trained on D_0 . Since the human instructor maintains a fixed driving strategy and doesn’t make critical mistakes, only a fraction of the state space is encountered while accumulating D_0 , leading to a not robust driving strategy. Second, we let the learned policy network, P_{θ_0} , to drive on its own. A human instructor then labels the states encountered by P_{θ_0} (i.e. assigns actions to them). These labeled states compose the training sample, $D_1 = \{(s_i, \text{human_label}(s_i))\}_i$ (in our experiments D_1 contained approximately 50,000 samples). We then train a policy network using $D = D_0 \cup D_1$, denoted by P_θ . The negative log-likelihood is used as a loss function for training both P_{θ_0} and P_θ ,

$$NLL(\theta, \mathcal{D}) = -\frac{1}{|\mathcal{D}|} \sum_{i=0}^{|\mathcal{D}|} \log P_\theta(a_i | s_i),$$

where P_θ is a policy network, which receives the state s and outputs a probability distribution on the optional actions, with parameters θ .

The performance of P_θ is significantly better when driving the training tracks (where training data is gathered) rather than unobserved tracks. This can reasonably be attributed to some degree of overfitting, which is not surprising given the relatively small number of training examples. During the RL procedure we will be able to improve this performance on unobserved test tracks. A detailed explanation of the performance evaluation procedure is given in Section 5, The parameters of P_θ were used to initialize the Q-network and the reward networks.



(a) The policy network's last layer output. (b) The policy network's architecture.

Figure 1: The policy network's architecture and its last layer's output. Each convolutional layer is followed by a RELU nonlinearity. A RELU nonlinearity is also applied on the first fully-connected layer. The numbers representing the convolutional layers and the Input layer are denoted for the number of channels \times height \times width. A softmax nonlinearity is applied on the last layer. The output of the last convolutional layer can be viewed as the low-dimensional representation, $F_1(s)$, of the raw signal.

3 Deep reward shaping

The problem of designing suitable reward functions to guide an agent to successfully learn a desired task is known as *reward shaping* (Laud, 2004). The idea is to define supplemental reward signals so as to make a RL problem easier to learn. The handcrafting of a reward function can be a complicated task, requiring experience and a fair amount of specific domain knowledge. Therefore, other methods for designing a reward function without the domain expertise requirement have been studied. In *Inverse RL* (IRL) a reward function is learned from expert demonstration (Abbeel & Ng, 2004). IRL algorithms rely on the fact that an expert demonstration implicitly encodes the reward function of the task at hand, and their goal is to recover a reward function which best explains the expert's behavior.

Daniel et al. (2014) proposed to learn a reward model in a supervised manner and use iterations between reward learning and reinforcement learning to continually improve the reward model and the agent’s policy. Their approach, which is based on manual feature generation by experts, has been applied in a robotic arm grasping task. The reward function is not learned from the raw states visited by the learner, but from some assumed to be known, low-dimensional feature representation of them. Constructing such low-dimensional representations usually requires some expert domain knowledge, making the learning of the reward function somewhat less advantageous.

Our reward shaping component for implementing the self-driving tasks learns the reward function directly from the raw image pixels. Since we don’t have access to the internal state of a simulator, defining a reward signal using the state parameters of the car (distances from the roadsides, angle with respect to the road, etc.) is impossible without explicit image processing. In this work, we devise and utilize a deep reward shaping network that is learned from a human driving instructor. The reward network, which is implemented with convolutional layers, maps a game state into the instantaneous reward, $r \in [-1, 1]$, corresponding to that state, $R_\theta : s \rightarrow r$. The driving instructor provides binary labeling for each state such that the reward network is a mapping from raw image pixels to $\{ \text{“good”}, \text{“bad”} \}$. The binary labeling of the human instructor can be viewed as a relatively easy task (for the human), which must continue until we are convinced that the reward model is sufficiently accurate.

We consider two different driving tasks: the easier task (Task 1) is to drive safely and smoothly, possibly using all lanes. The second task (Task 2) is similar but requires that the agent drive only in a predefined lane. For these two tasks we train two reward networks, denoted by $R_\theta^{general}$ and R_θ^{lane} , which share the same architecture. Their architecture is identical to that of the policy network, except for the final fully connected layer that now has one node instead of three. A hyperbolic tangent (tanh) activation function is applied on the last layer to receive output in the range $[-1, 1]$. The learned parameters of the first layers of P_θ (the parameters that correspond to $F_1(s)$) are used to initialize both reward networks. The reward networks are trained with data recorded from a human instructor in a supervised learning procedure, with different datasets (including labeling) for each reward network. The MSE loss objective is used to train both networks,

$$\mathbb{E}_{s, l \sim \mathcal{D}} \left[(l - R_\theta(s))^2 \right],$$

where $l \in \{-1, 1\}$ is the corresponding label given by the human instructor.

Task 1: A reward model for general driving. The reward network $R_\theta^{general}$ is trained to give high rewards to “good” states where the car is on the road facing the correct direction (whether it is in the correct lane is not important), as judged by the human instructor. Those “good” states were labeled one. All other states were labeled minus one.² The training of $R_\theta^{general}$

²To ease the human instruction efforts all of the one-labeled samples were reused from the imitation training data.

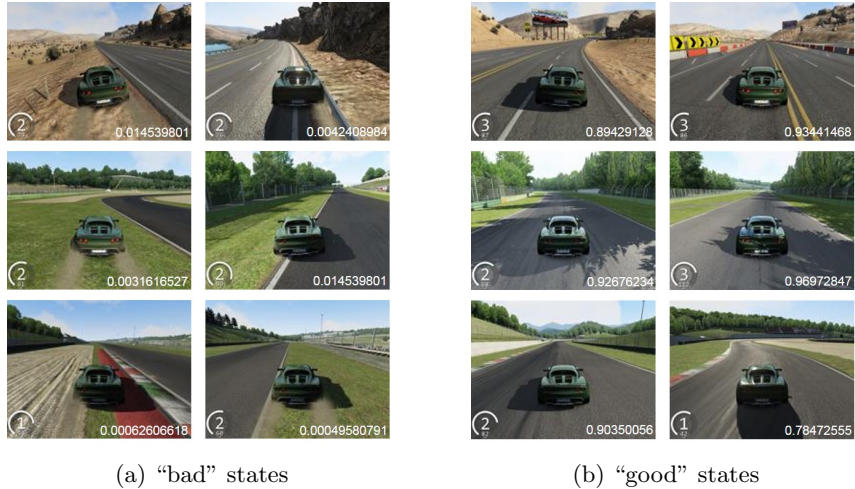


Figure 2: $R_{\theta}^{general}$ outputs high reward for being in “good” states according to the instructor (b), and low reward to “bad” states (a). The reward is the number located in the lower-right corner of each frame.

continued until a statistical test indicated that this network provides accurate reward over test data. We note that the binary learning task of the reward network is considerably easier than the imitation task.

Task 2: A reward model for lane-constrained driving. During the imitation learning stage the human instructor ignored the lane marks on the road (some of the tracks do not contain lane marks at all), but in Task 2 we considered a more complex application whereby the agent must drive in a designated lane only (the second lane from the right on a 4-lane road). To this end we trained a reward network R_{θ}^{lane} , to give high reward only for states where the car was in that specific lane. R_{θ}^{lane} was trained with approximately 30,000 new samples corresponding to roughly one driving hour. States in which the car was in the designated lane were labeled one, and all other states were labeled minus one. A training curve of R_{θ}^{lane} is presented in Figure 3, showing that a 97% validation accuracy was achieved with early stopping. An example of the output of R_{θ}^{lane} over different states is given in Figure 4, showing very high reward to the designated lane.

4 Reinforcement learning using DDQN

In this section we assume basic familiarity with reinforcement learning; see, e.g., (Sutton & Barto, 1998). In the final RL stage, we utilize the already trained reward networks and apply them within a standard RL method. Recalling that performance is measured in terms of the learned reward model, the main goal of this stage is to achieve a significantly better performance level than the one

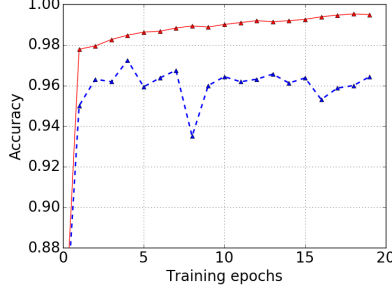


Figure 3: Training curve of R_{θ}^{lane} . The red solid line represent training performance and the blue dashed line represent validation performance. R_{θ}^{lane} was trained on data generated from the “Black Cat County” track.



Figure 4: R_{θ}^{lane} outputs high reward to driving in the second lane from the right (b), and low reward to other lanes (a). The reward is the number located in the lower-right corner of each frame.

achieved in the mimicry stage by enabling the agent to teach itself. In other words, starting with a policy π_0 , we would like to apply an RL algorithm to learn a policy π^* , which is optimal with respect to the expected (discounted) reward received from the reward network.

While any RL method can in principle be used in our scheme, we utilize a variant of the Q-learning algorithm (Watkins & Dayan, 1992), which aims to find the *optimal action-value function*, denoted by $Q^*(s, a)$, and defined as the expected (discounted) reward after taking action a in state s and following the optimal policy π^* thereafter. The true value of taking action a in state s under policy π is

$$Q^{\pi}(s, a) = \mathbb{E} [r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \mid s_t = s, a_t = a; \pi],$$

where $\gamma \in [0, 1]$ is a fixed discount factor and r is the guidance reward signal. Given the optimal action-value function, which is defined to be $Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$, the optimal policy π^* can be simply derived by taking the action with the highest action-value function in each state. Dealing with a very large state space of images, we approximate the optimal action-value function using a *deep Q-network* (DQN) with parameters θ :

$$Q_{\theta}(s, a) \approx Q^*(s, a).$$

A deep Q-network is a neural network that for a given state outputs a vector of action values.

We used the DDQN algorithm with replay memory and target values calculated from parameters of the previous iteration, as in the work of Mnih et al. (2015). The loss function we used is therefore,

$$L(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{B}} \left[\left(r + \gamma Q_{\tilde{\theta}}(s', \operatorname{argmax}_{a'} Q_{\theta}(s', a')) - Q_{\theta}(s, a) \right)^2 \right], \quad (1)$$

where (s, a, r, s') are samples taken from the replay memory buffer \mathcal{B} , and $\tilde{\theta}$ are the Q-network’s parameters from the previous iteration. In our applications, the reward received after taking action a in state s is the instantaneous reward obtained from the reward network for state s (see Section 3). We set γ to be 0.9 and used the ADAM (Kingma & Ba, 2014) method for stochastic gradient optimization.

We trained a Q-network, denoted by Q_{θ} , that has the same architecture as P_{θ} except for the final softmax nonlinearity, which was removed. The Q-network’s parameters were initialized from those of the policy network. We wished, on the one hand, to maintain the policy network’s final convolutional layer’s informative state representations. On the other hand we also wished to allow the Q-network to output values compatible with Q-learning temporal difference targets. Therefore, an initial policy evaluation step was performed, where we let P_{θ} drive, and updated the parameters of only the last two fully-connected layers of Q_{θ} while fixing all of its other parameters. This step can be viewed as learning a critic (and more precisely, only the last two layers of the critic’s network) to estimate an action-value function while fixing the acting policy. After this policy evaluation step we started the RL algorithm using Q_{θ} , allowing all of its parameters to be updated. An alternative way to initialize the Q-network’s last two fully-connected layers was also considered. Here, in the first (approximately) 12 hours of the RL procedure (using the Q-network to output actions), we allowed only the parameters of the two final fully connected layers to be updated. All the other parameters were fixed. After these initial 12 hours, we allowed the fixed parameters to be updated as well. Full details of the experiments are given in Section 5.

5 Experimental setting and technicalities

RL experiments. Q_{θ} was independently trained twice to perform both tasks 1 and 2. For Task 1 we used $R_{\theta}^{general}$ on the “Mugello” track. For Task 2 we used R_{θ}^{lane} on the “Black Cat County” track. The Q-network’s parameters were initialized from those of the policy network for both tasks. Game restarts were performed when the agent had zero speed (e.g., stuck against some obstacle) or when it drove in the wrong direction (both of these situations were detected through simple image processing). The agent operated with an ϵ -greedy policy ($\epsilon = 0.05$). We trained each task for a total of 4 million frames (that is,

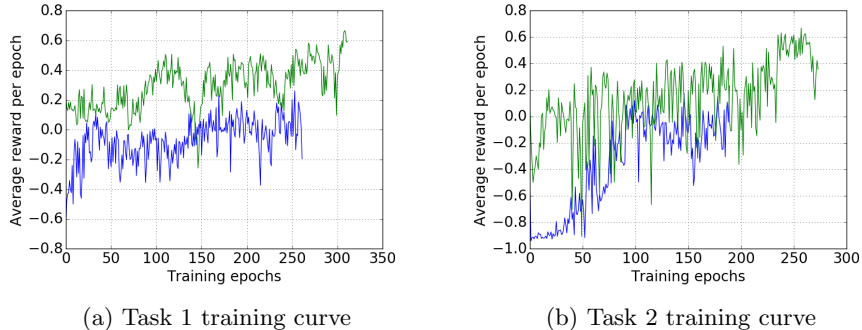


Figure 5: Agent’s average reward training curves. An epoch is defined to be (approximately) 15,000 sample frames. Each point in the graphs is the average reward achieved per epoch. (a) Training curves for Task 1 using the reward network $R_{\theta}^{general}$. The Q-network is initialized with the IL policy network parameters (green), and with random initialization (blue); (b) Training curves for Task 2 using the reward network R_{θ}^{lane} , with its Q-network initialized with the IL policy network parameters (green), and with random initialization (blue).

around 5 days of game experience in total) and used a replay memory of the 5,000 most recent frames. We were able to successfully train the Q-network on both tasks, as can be seen from the results, which are presented in Figure 5. As can be seen in Figure 5, initializing the Q-network’s parameters with those learned in the mimicking stage resulted in better overall driving performance than that achieved when initializing the Q-network’s parameters randomly. The advantage of such *informed initialization* (using the IL policy network’s parameters) over the random initialization is most significant at the early stages of the reinforcement learning, and circumvented critical driving mistakes that were observed with a random initialization. In Task 2 the agent outperformed the human instructor’s initial demonstration; namely, it received superior reward from R_{θ}^{lane} (the instructor’s average reward per epoch was below zero).

Technical implications of not having access to the simulator’s internal state. While performing the RL algorithm, all of the following operations are executed online: image capturing, image pre-processing, reward prediction by the reward network, action prediction by the Q-network, the Q-network parameter updates, etc. On average, the serial execution of these operations takes 7 ms. Therefore, the agent must always act under 7 ms latency (from the time the image is captured to the time the corresponding steering key is pressed). During this gap, the previous key continues to be pressed until a new one is received. This added difficulty can be avoided when using a simulator. But most importantly, our inability to apply the RL stage in a super-fast simulation speed limits the number of training epochs we can afford to conduct.

Image preprocessing. The images were resized from 1024×768 to 192×144 and were not converted to gray scale. The pixel values were scaled to be in

the range $[0, 1]$. Pixels corresponding to the speed indicator were set to zero in order to guarantee that the agent doesn't use the speed information during the learning process. In all our applications through this work, each input instance consisted of two sequential images with a 0.5-second gap between them.

Work environment. We used the Theano-based Lasagne library for implementing the neural networks. Two GeForce GTX TITAN X GPUs were utilized during the RL experiments; one was used to run the racing game and the other, to train the networks and predict the rewards and actions.

Data generation and network training technicalities. The human demonstrator played the game using a "racing" strategy: driving as fast as possible while ignoring the lane marks and trying to avoid accidents. Training data for the policy and reward networks was collected from the tracks: 'Black Cat County', 'Imola' and 'Nürburgring gp'. Sample images of the tracks we used are presented in Figure 6. Using a Python environment, screen images were recorded every 0.1 seconds while the human demonstrator played the game. Keyboard keys pressed by the demonstrator were also recorded. 80% of the recorded samples were used as training data and the remaining 20% as validation data. We used the ADAM stochastic optimization method with dropout for regularization (Srivastava et al., 2014). The network parameters that achieved best validation accuracy were chosen.

Performance evaluation. Without a natural performance evaluation measure for driving skills and without access to the internal state of the game, our performance evaluation procedure was based on the accumulated average reward achieved by the agent. Better driving means higher (discounted) accumulated average reward and vice versa. Assuming that our learned reward model reliably reflects the concept of "good driving" (with or without lane constraints), the accumulated average reward is a sound performance measure for the required task.

Other technicalities. All of the experiments were conducted without any other vehicles on the road. We used the 'Lotus Elise SC' car model in all of our experiments. Focusing only on the steering control problem, we eliminated the accelerator/brakes control variability by using a "cruise control" behavior whereby the car's speed was set to 50 kmh. This was achieved by extracting the car's speed from the game image and applying a simple hand-crafted controller.

6 Concluding remarks

We presented a multi-stage generic framework, which exploits a natural synergy between several learning principles, for training an agent to perform a complex task. We expect the proposed framework to be useful in various application domains, and have demonstrated its strength on autonomous highway steering problems. Our solution relied on recent deep representational methods such as CNNs to successfully implement all three components of the proposed framework. First, we train a CNN agent to imitate a human demonstrator aiming at achieving a basic initial driving policy. Second, an instantaneous reward

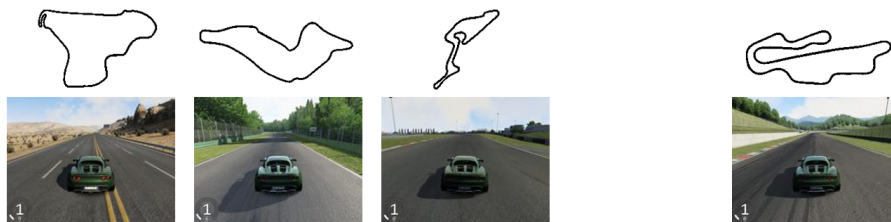


Figure 6: Training and test tracks with sample frames. The three left images are the training tracks (presented from left to right): “Black Cat County”, “Imola” and “Nürburgring gp”. The right most image is the test track “Mugello”. The “Black Cat County” track (left most) is the only one with lane marks.

network is trained from human instructions. Third, the agent uses the learned reward network’s output as a guidance signal in a RL procedure. We note that the proposed implementation is scalable to any computerized driving game and even to real-world driving.

The main strength of the proposed scheme stems from leveraging the weak supervision abilities of a (human) instructor, who, while unable to perform well herself at the required task, can provide coherent and learnable instantaneous reward signals to the computerized trainee. This leveraging effect clearly occurred in our self-steering example, where single-lane driving demonstrations (by the instructor) were not included in the imitation stage (and moreover, most of the training tracks do not even include lane marks). Yet, the agent quite easily learned to drive in a single designated lane after receiving one hour instruction session followed by self reinforcement learning (without the instructor).

We believe that our self-steering example can be improved in various ways, and in fact, we have not attempted to optimize it. For example, using a steering wheel controller rather than a keyboard, should improve the learning process and the final driving performance. Similarly, the raw images, used as inputs to all our neural networks, were not optimized to the best visual region of interest (ROI), and contain a considerable amount of information irrelevant to the task. It would also be interesting to extend our solution to handle the accelerator/brakes controllers.

Both effective acquisition of instantaneous reward from an instructor and accurate modeling of the reward function are required for a successful application of the proposed framework. While in our driving example the reward models were easily constructed, creating these models for highly complex tasks is expected to be challenging in terms of both model capacity and the development of effective methodologies for interaction with the instructor. We anticipate that harnessing the supervision abilities of a (human) instructor, for the purpose of learning an effective reward model, will become a critical building block in creating robots capable of adjusting themselves to human needs.

References

- P. Abbeel and A. Y. Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pp. 1, 2004.
- B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.
- C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning Affordance for Direct Perception in Autonomous Driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2722–2730, 2015.
- Adam Coates, Honglak Lee, and Andrew Y Ng. An analysis of single-layer networks in unsupervised feature learning. *Ann Arbor*, 1001(48109):2, 2010.
- C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters. Active Reward Learning. In *Proceedings of Robotics Science & Systems*, 2014.
- H. Van Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.
- H. Van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- G. M. Hayes and J. Demiris. A Robot Controller Using Learning by Imitation. Technical report, University of Edinburgh, Department of Artificial Intelligence, 1994.
- John Illingworth and Josef Kittler. A survey of the Hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.
- D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- J. Kober, J. A. Bagnell, and J. Peters. Reinforcement Learning in Robotics: A survey. *The International Journal of Robotics Research*, 2013.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.
- A. D. Laud. *Theory and Application of Reward Shaping in Reinforcement Learning*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.
- D. Loiacono, A. Prete, P. L. Lanzi, and L. Cardamone. Learning to Overtake in TORCS Using Simple Reinforcement Learning. In *IEEE Congress on Evolutionary Computation*, pp. 1–8, 2010.

- V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, A. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, pp. 529–533, 2015.
- J. Munoz, G. Gutierrez, and A. Sanchis. Controller for TORCS created by imitation. In *IEEE Symposium on Computational Intelligence and Games*, pp. 271–278, 2009.
- A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pp. 278–287, 1999.
- Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 689–696, 2011.
- S. Ross, G. J. Gordon, and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *International Conference on Artificial Intelligence and Statistics*, volume 1, pp. 6, 2011.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press Cambridge, 1998.
- M. E. Taylor, H. B. Suay, and S. Chernova. Using Human Demonstrations to Improve Reinforcement Learning. In *AAAI Spring Symposium: Help Me Help You: Bridging the Gaps in Human-Agent Collaboration*, 2011.
- C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, pp. 279–292, 1992.
- B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner. TORCS, the open racing car simulator. *Software available at <http://torcs.sourceforge.net>*, 2000.
- J. Zhang and K. Cho. Query-Efficient Imitation Learning for End-to-End Autonomous Driving. *arXiv preprint:1605.06450*, 2016.